Journal of Computational and Applied Mathematics 23 (1988) 87–98
North-Holland

Algorithm 30

Fast numerical solution of weakly singular Volterra integral equations

E. HAIRER
Université de Genève, Departement de Mathematique, Rue du Lièvre 2-4, Case Postale 240, CH-1211 Genève 24, Switzerland

Ch. LUBICH
Universität Innsbruck, Institut für Mathematik und Geometrie, Technikerstraße 13, A-6020 Innsbruck, Austria

M. SCHLICHTE
Universität Karlsruhe, Numerische Strömungsmechanik, Rechenzentrum, Zirkel 2, D-7500 Karlsruhe, Fed. Rep. Germany

Received 20 June 1986

Abstract: We give a FORTRAN program for the numerical solution of Abel–Volterra convolution equations of the second and first kind. We use a fractional linear multistep method $(\mathrm{BDF4})^{1/2}$ and use Fast Fourier transform techniques to exploit the convolution structure. A detailed description of the computation of the quadrature weights using FFT is given. Numerical examples are included.

Keywords: Abel–Volterra convolution equation, fractional multistep method, Fast Fourier Transform.

1. Introduction

In this paper we give a FORTRAN program for the numerical solution of scalar, nonlinear, weakly singular Volterra integral equations of the form

$$y(t) = f(t) + \frac{1}{\sqrt{\pi}} \int_0^t (t-s)^{-1/2} k(t-s) g(s,\ y(s))\ \mathrm{d}s, \quad 0 \leqslant t \leqslant T. \tag{1}$$

It is assumed that k is sufficiently smooth, and that $f(t) = F(\sqrt{t})$ and $g(t,\ y) = G(\sqrt{t},\ y)$ for some smooth functions F and G. The solution $y(t)$ is then again a smooth function of $\sqrt{t}$.

The algorithm is based on the fourth order backward differentiation formula to power $1/2$, whose convergence and stability properties have been studied in [6], see also [5], [1] and [7]. The convolution structure in (1) is exploited to yield numerical approximations at N equidistant

grid-points using $O(N)$ f-, k-, and g-evaluations and $O(N(\log N)^2)$ arithmetical operations, the latter being achieved by Fast Fourier Transform techniques as in [2].

In Section 2 we review the discretization of (1) by fractional powers of backward differentiation methods. In Section 3 we describe in detail the computation of the quadrature weights by Fast fourier Transform techniques. In Section 4 we give a description of the algorithm for solving the discretized convolution equation (1). Section 5 explains the FORTRAN subroutine, and we present some numerical examples in Section 6.

We remark that the algorithm can easily be modified (see Section 5) to be applicable to equations of the first kind

$$0 = f(t) + \frac{1}{\sqrt{\pi}} \int_0^t (t-s)^{-1/2} k(t-s) g(s, y(s)) \, ds, \quad 0 \leqslant t \leqslant T, \tag{2}$$

where we assume additionally $k(0) \neq 0$ and $\partial g / \partial y \neq 0$. Convergence for this case is studied in [8].

2. Fractional powers of backward differentiation methods

Equation (1) is discretized by a quadrature method ($n = 0, 1, \ldots, N + 2p - 2$)

$$y_n = f_n + \sqrt{h} \sum_{j=2p-1}^{n} \omega_{n-j} k(t_n - t_j) g(t_j, y_j), \tag{3}$$

where

$$f_n = f(t_n) + \sqrt{h} \sum_{j=0}^{2p-2} w_{nj} k(t_n - t_j) g(t_j, y_j), \tag{4}$$

$h > 0$ is the stepsize, and $t_n = nh$. The convolution quadrature weights ω_n are the coefficients of the power series

$$\sum_{j=0}^{\infty} \omega_j \zeta^j = \left(\sum_{k=1}^{p} \frac{1}{k} (1-\zeta)^k \right)^{-1/2}, \tag{5}$$

where p ($1 \leqslant p \leqslant 6$) is the order of the underlying backward differentiation formula. The starting quadrature weights w_{nj} are chosen in such a way that for $k(t) \equiv 1$ the quadrature method integrates the functions $t^{q/2}$ for $q = 0, 1, \ldots, 2p-2$ exactly. Equivalently, for each n the weights w_{nj} are the solution of the Vandermonde-type linear system

$$\sum_{j=0}^{2p-2} w_{nj} j^{q/2} = \frac{\Gamma(q/2+1)}{\Gamma(q/2+3/2)} n^{q/2+1/2} - \sum_{j=2p-1}^{n} \omega_{n-j} j^{q/2}, \quad q = 0, 1, \ldots, 2p-2. \tag{6}$$

Under the smoothness assumptions of Section 1, it is shown in [6] that the above method is convergent of order p, i.e.,

$$y_n - y(t_n) = O(h^p) \quad \text{uniformly for } 0 \leqslant nh \leqslant T.$$

3. Computation of quadrature weights

For the computation of the weights ω_j defined by (5) we use Newton's method for formal power series (see [3, §13.9]):

The power series

$$\omega(\zeta) = \sum_{j=0}^{\infty} \omega_j \zeta^j$$

satisfies by (5)

$$\omega(\zeta)^{-2} = q(\zeta) \quad \text{with } q(\zeta) = \sum_{k=1}^{p} \frac{1}{k}(1-\zeta)^k,$$

or, writing this more formally,

$$F(\omega(\zeta)) = 0 \quad \text{with } F(w) = w^{-2} - q(\zeta).$$

For a power series $a(\zeta) = \Sigma_0^\infty a_j \zeta^j$, let $[a(\zeta)]_n = \Sigma_0^{n-1} a_j \zeta^j$ denote the truncated series. With this notation, Newton's method for power series reads:

$$\omega^{(0)}(\zeta) = \omega_0 \quad \text{is determined by } [F(\omega_0)]_1 = 0,$$

$$\omega^{(m+1)}(\zeta) = \left[\omega^{(m)}(\zeta) - F'(\omega^{(m)}(\zeta))^{-1} F(\omega^{(m)}(\zeta))\right]_{2^{m+1}}.$$

It is then known that

$$\omega^{(m)}(\zeta) = [\omega(\zeta)]_{2^m} = \sum_{j=0}^{2^m-1} \omega_j \zeta^j \quad \text{for every } m \geqslant 0.$$

For our special case $F(w) = w^{-2} - q(\zeta)$ the algorithm becomes

$$\omega^{(0)}(\zeta) = \omega_0 = q(0)^{-1/2}, \tag{7}$$

$$\omega^{(m+1)}(\zeta) = \left[\tfrac{3}{2}\omega^{(m)}(\zeta) - \tfrac{1}{2}(\omega^{(m)}(\zeta))^3 q(\zeta)\right]_{2^{m+1}}.$$

This algorithm requires only to compute the coefficients of the product of polynomials. This can be done efficiently by Fast Fourier Transforms (see [3, §13.8]). The weights $\omega_0, \omega_1, \ldots, \omega_{N-1}$ ($N = 2^m$) are thus computed in $O(N \log N)$ arithmetical operations.

Once the weights ω_j have been obtained, we compute the right-hand sides of (6) simultaneously for all n, using Fast Fourier Transforms for the discrete convolution. This requires another $O(N \log N)$ arithmetical operations. The starting quadrature weights w_{nj} are then readily obtained from the Vandermonde system (6) (whose matrix does not depend on n).

We remark that there is cancellation of digits in the evaluation of the right-hand side of (6), and the Vandermonde system is ill-conditioned. Hence the weights w_{nj} are computed with possibly low accuracy. However, the residual in (6) is small, which is all that matters for the accuracy of the quadrature method.

4. Solution of the discrete convolution equation

For the computation of the approximate solution values y_n we proceed as follows: Firstly, we have $y_0 = y(0) = f(0)$. The values $y_1, \dots, y_{2p-2}$ are then obtained from

$$y_n = f(t_n) + \sqrt{h} \sum_{j=0}^{2p-2} w_{nj} k(t_n - t_j) g(t_j, y_j) \quad \text{for } n = 1, \dots, 2p-2. \tag{8}$$

This nonlinear system of equations is solved by Newton's method, using the starting values $y_n^{(0)} = f(t_n)$. With these approximations we compute the values f_n, given by (4), for $n = 2p-1, \dots, N+2p-2$. Upon letting

$$\tilde{t}_n = t_{n+2p-1}, \qquad \tilde{f}_n = f_{n+2p-1}, \qquad \tilde{y}_n = y_{n+2p-1},$$

and

$$\kappa_j = \sqrt{h}\, \omega_j k(t_j),$$

formula (3) becomes the discrete convolution equation

$$\tilde{y}_n = \tilde{f}_n + \sum_{j=0}^{n} \kappa_{n-j} g(\tilde{t}_j, \tilde{y}_j), \quad n = 0, 1, \dots, N-1. \tag{9}$$

The values $\tilde{y}_0, \tilde{y}_1, \dots$ can be computed one after the other as solution of a scalar nonlinear equation by Newton's method. An efficient computation is organized as in [2]:

From (9) we compute the first $r = 2^5$ values $\tilde{y}_0, \dots, \tilde{y}_{r-1}$. Then also $\gamma_j = g(\tilde{t}_j, \tilde{y}_j)$ for $j = 0, 1, \dots, r-1$ are known, and we compute the discrete convolution

$$\sum_{j=0}^{r-1} \kappa_{i-j} \gamma_j \quad \text{for } i = r, r+1, \dots, 2r-1, \tag{10}$$

using FFT. The values $\tilde{y}_r, \dots, \tilde{y}_{2r-1}$ are then obtained from (9), using the known partial sums (10). Next we compute the convolution $\sum_{j=0}^{2r-1} \kappa_{i-j} \gamma_j$ for $i = 2r, \dots, 4r-1$, using again FFT. The continuation of this process is illustrated in Fig. 1. There every square represents the evaluation

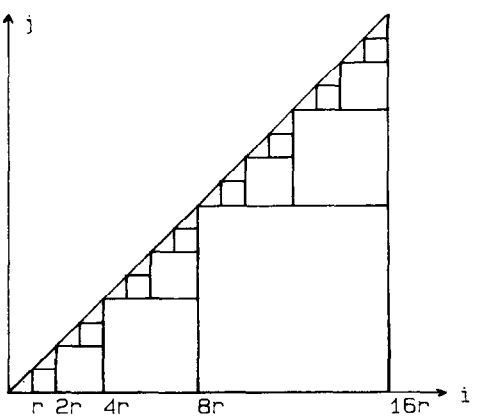

Fig. 1. Organization of the computation.

of one convolution by FFT. The solution values $\tilde{y}_0, \ldots, \tilde{y}_{N-1}$ are thereby obtained in $O(N(\log N)^2)$ arithmetical operations.

5. Description of the FORTRAN subroutine ABELFT

The algorithm described above has been implemented as a FORTRAN program ABELFT in double precision. The list of input and output parameters is given in the program. A driver program is included below. Let us give here the dependence chart of the required subroutines (Fig. 2). The subroutines C05NBF, C05AJF (solution of nonlinear equations), C06EAF, C06EBF (Fast Fourier Transforms), and F03AFF, F04AJF (linear systems solver) are those of the NAG library. The functions FCN, FKER, GSY are the implementations of the functions f, k, and g of (1) and have to be supplied by the user. The subroutine VALINT specifies the nonlinear system of equations (8), the subroutine STEP specifies the nonlinear equation for forward integration (9). The subroutine WBDF4 computes the weights ω_j and w_{nj} given by (5) and (6) with $p = 4$. It uses the subroutine CONVOL, which evaluates the Cauchy product of two sequences.

The subroutine WBDF4 is called only once at the start of ABELFT. This call can be taken out of the subroutine. This is recommended when ABELFT is to be called more than once.

For the solution of the first-kind equation (2) only two FORTRAN statements in subroutines VALINT and STEP have to be altered, as indicated there.

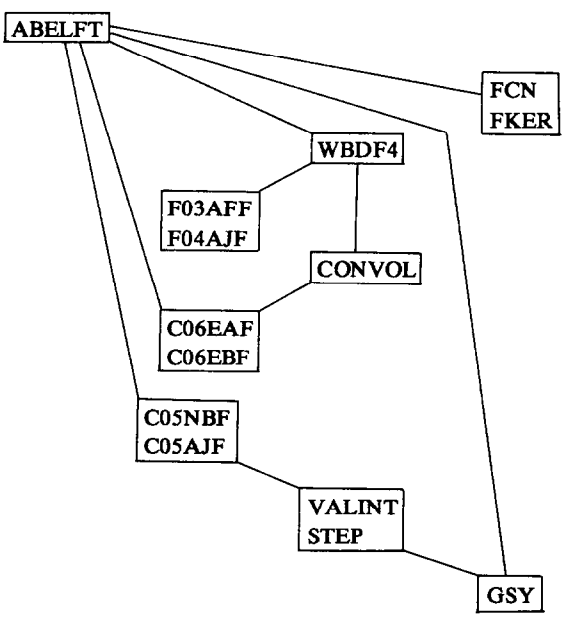

Fig. 2. Dependence chart for ABELFT.

6. Numerical experiments

In order to verify numerically that the method has convergence order 4 also for nonsmooth solutions, we have applied the program to the equation

$$y(t) = f(t) - \frac{1}{\sqrt{\pi}} \int_0^t (t-s)^{-1/2} y(s)\, ds, \tag{11}$$

where $f(t)$ is chosen such that the solution is given by

$$y(t) = \sum_{j=0}^{11} \frac{t^{j/2}}{\Gamma(j/2+1)}.$$

The global error, divided by h^4,

$$(y_n - y(t_n))/h^4, \tag{12}$$

is plotted for several stepsizes in Fig. 3. One nicely observes the uniform convergence to a function $e(t)$, which illustrates that the error is of the form

$$y_n - y(t_n) = e(t_n)h^4 + o(h^4).$$

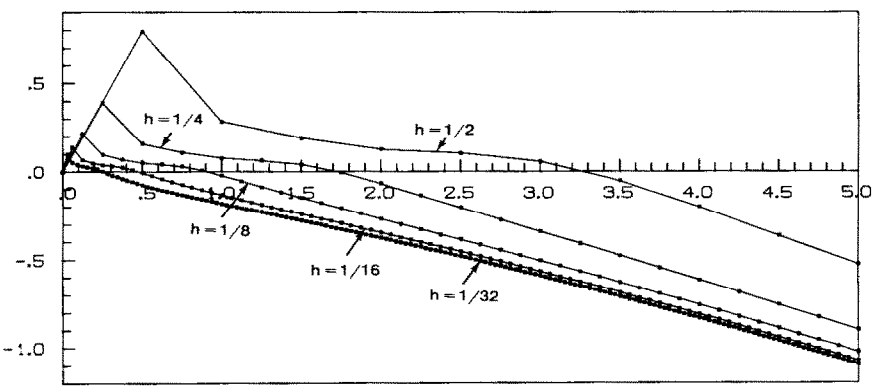

Fig. 3. Error divided by h^4 for equation (11).

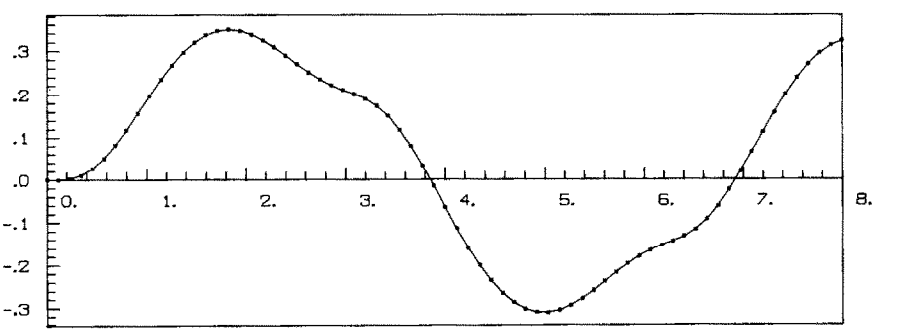

Fig. 4. Solution of (13).

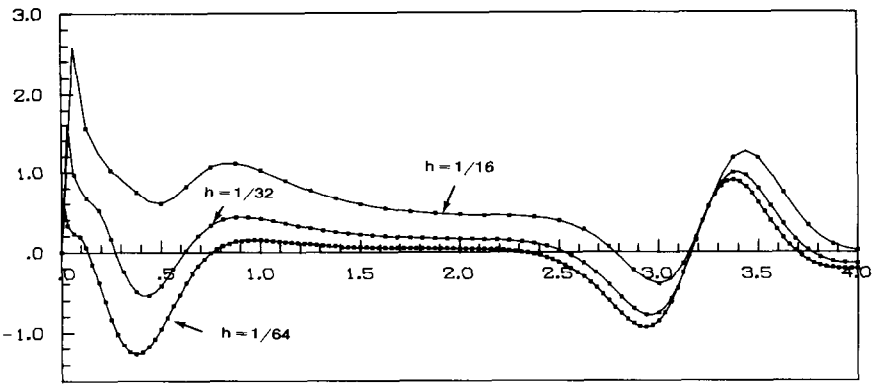

Fig. 5. Error divided by h^4 for equation (13).

As a second example we have considered the equation

$$y(t) = -\frac{1}{\sqrt{\pi}} \int_0^t (t-s)^{-1/2} (y(s) - \sin s)^3 \, ds, \tag{13}$$

which arises in the theory of superfluidity [4]. Its solution as obtained by ABELFT is plotted in Fig. 4. Also for this nonlinear problem we have investigated the divided error (12) (see Fig. 5), and one observes again convergence order 4.

References

[1] E. Hairer and Ch. Lubich, An extension of ODE-methods to Volterra integral equations. *Proceedings of "Numerische Behandlung von Differentialgleichungen"*, Halle (DDR), 1985, Teubner-Texte zur Mathematik (Teubner, Leipzig, 1985).

[2] E. Hairer, Ch. Lubich and M. Schlichte, Fast numerical solution of nonlinear Volterra convolution equations, *SIAM J. Sci. Stat. Comput.* **6** (1985) 532–541.

[3] P. Henrici, *Applied and Computational Complex Analysis, Vol. 3* (Wiley, New York, 1986).

[4] N. Levinson, A nonlinear Volterra equation arising in the theory of superfluidity, *J. Math. Anal. Appl.* **1** (1960) 1–11.

[5] Ch. Lubich, Discretized fractional calculus, *SIAM J. Math. Anal.* **17** (1986) 704–719.

[6] Ch. Lubich, Fractional linear multistep methods for Abel-Volterra integral equations of the second kind, *Math. Comput.* **45** (1985) 463–469.

[7] Ch. Lubich, On the numerical solution of Volterra equations with unbounded nonlinearity, *J. Integral Equations* **10** (1985) 175–183.

[8] Ch. Lubich, Fractional linear multistep methods for Abel-Volterra integral equations of the first kind, *IMA J. Numer. Anal.* **7** (1987) 97–106.

```
      SUBROUTINE ABELFT(M,XEND,Y,FK,WORK,W,WS,NWS)
C -------------------------------------------------------------------
C     NUMERICAL SOLUTION OF A SCALAR, NONLINEAR, WEAKLY SINGULAR
C     SECOND KIND ABEL INTEGRAL EQUATION OF THE TYPE
C                                  X
C        Y(X) = F(X) + 1/SQRT(PI) I (X-S)**-1/2*K(X-S)*G(S,Y(S)) DS
C                                  0
C     THE ALGORITHM IS BASED ON THE FRACTIONAL BDF-METHOD OF ORDER 4.
C     (K IS ASSUMED TO BE SMOOTH, F A SMOOTH FUNCTION OF SQRT(X),
C     G A SMOOTH FUNCTION OF SQRT(X) AND Y).
C
C     EXTERNAL FUNCTIONS (TO BE SUPPLIED BY THE USER)
C     -----------------
C     FCN(X)     FUNCTION F
C     FKER(X)    KERNEL FUNCTION K(X)
C     GSY(S,Y)   FUNCTION G
C
C     INPUT PARAMETERS
C     ----------------
C     M          2**M+7 IS THE NUMBER OF SOLUTION POINTS;
C                DENOTE   N=2**M
C     XEND       ENDPOINT OF INTEGRATION
C     FK(2*N)    WORKSPACE (KERNEL EVALUATIONS AND THEIR FFT'S)
C     WORK(2*N)  WORKSPACE (GSY-VALUES)
C     W(N)       QUADRATURE WEIGHTS (INTERNALLY COMPUTED BY WBDF)
C     WS(7,N+6)  STARTING WEIGHTS (INTERNALLY COMPUTED BY WBDF)
C     NWS        FIRST DIMENSION OF ARRAY WS(NWS,1)
C
C     OUTPUT PARAMETERS
C     -----------------
C     Y(N+7)     FOR I=1,...,N+7  Y(I) CONTAINS THE NUMERICAL
C                SOLUTION AT XI=(I-1)*H  WHERE H=XEND/(N+6)
C -------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      EXTERNAL VALINT,STEP
      DIMENSION Y(1),FK(1),WORK(1),W(1),WS(NWS,1),GS(7),WA(93)
      COMMON /SYST/A(6,6),H,B(6)
      COMMON /FORWAR/C1,F1,T1
      DATA MR/5/,KORD/4/,EPS/1.D-10/,ETA/1.D-12/,LWA/93/
C --- INITIAL PREPARATIONS ---
      KSTEP=2*KORD-1
      KM1=KSTEP-1
      MR=MIN0(MR,M)
      N=2**M
      NR=2**MR
      CALL WBDF4(W,WS,NWS,WORK,FK,N)
      H=XEND/DFLOAT(N+KM1)
      SQH=DSQRT(H)
C --- COMPUTE FUNCTION- AND KERNEL-VALUES ---
      DO 5 I=1,N+KSTEP
      XI=(I-1)*H
      Y(I)=FCN(XI)
    5 FK(I+KM1-1)=FKER(XI)
      DO 7 I=1,KM1-1
    7 FK(I)=FKER((I-KM1)*H)
C --- SOLVE NON-LINEAR SYSTEM FOR INITIAL VALUES ---
      GS(1)=GSY(0.D0,Y(1))*SQH
      DO 10 I=1,KM1
      B(I)=WS(1,I)*FK(I+KM1)*GS(1)+Y(I+1)
      DO 10 J=1,KM1
   10 A(I,J)=WS(J+1,I)*FK(I-J+KM1)*SQH
      IFAIL=0
      CALL C05NBF(VALINT,KM1,Y(2),GS(2),EPS,WA,LWA,IFAIL)
C --- COMPUTATION OF INHOMOGENITY ---
```

```
      DO 15 I=2,KSTEP
 15   GS(I)=GSY((I-1)*H,Y(I))*SQH
      DO 25 I=KSTEP+1,N+KSTEP
      SUM=0.D0
      DO 20 J=1,KSTEP
 20   SUM=SUM+WS(J,I-1)*GS(J)*FK(I-J+KM1)
 25   Y(I)=SUM+Y(I)
C --- PREPARATION FOR FORWARD INTEGRATION ---
      C1=SQH*W(1)*FK(KM1)
      DO 40 I=1,N-1
 40   FK(I)=SQH*W(I+1)*FK(I+KM1)
C --- COMPUTE THE FFT'S OF THE KERNEL AND STORE THEM IN FK ---
      FK(N)=0.D0
      ISF=0
      IF (M.GT.MR) THEN
         ISF=N
         IF (M.GT.MR+1) THEN
            DO 47 I=2,M-MR
            N2I=N/2**(I-1)
            DO 46 J=1,N2I-1
 46         FK(ISF+J)=FK(J)
            FK(ISF+N2I)=0.D0
            IFAIL=0
            CALL C06EAF(FK(ISF+1),N2I,IFAIL)
 47         ISF=ISF+N2I
         END IF
         DO 48 J=1,NR
 48      FK(ISF+J)=FK(J)
         CALL C06EAF(FK,N,IFAIL)
      END IF
C --- FORWARD INTEGRATION ---
      DO 80 ISTEP=1,N/NR
      IPOS=(ISTEP-1)*NR
      KSPOS=KSTEP+IPOS
C ------ NEWTON METHOD ALONG THE DIAGONAL ---
      DO 55 I=1,NR
      T1=(KM1+IPOS+I)*H
      F1=Y(KSPOS+I)
      SOL=Y(KSPOS+I-1)
      IFAIL=0
      CALL C05AJF(SOL,EPS,ETA,STEP,100,IFAIL)
      Y(KSPOS+I)=SOL
C ------ STORE THE GSY(S,Y)-VALUES IN SECOND HALF OF WORK ---
      NIPOS=N+IPOS+I
      WORK(NIPOS)=GSY(T1,SOL)
      IF(I.LT.NR) THEN
         DO 50 J=I+1,NR
 50      Y(KSPOS+J)=Y(KSPOS+J)+WORK(NIPOS)*FK(ISF+J-I)
      END IF
 55   CONTINUE
      IF (IPOS+NR.EQ.N) RETURN
C ------ COMPUTE SIZE OF FFT-SQUARES ---
      IFK=0
      DO 60 J=1,ISTEP
      IFL=2**(J-1)
      IFK=IFK+IFL
 60   IF (MOD(ISTEP/IFL,2).EQ.1) GOTO 65
 65   IFK=ISF-2*NR*IFK
C ------ COMPUTE THE LAG IN THE FFT-SQUARE ---
      NRIFL=NR*IFL
      NRIFL2=NRIFL*2
      SQN=DSQRT(DFLOAT(NRIFL2))
      DO 70 I=1,NRIFL
      NIPOS=N+IPOS+I+NR-NRIFL
```

```
      WORK(I)=WORK(NIPOS)*SQN
   70 WORK(NRIFL+I)=0.D0
      CALL C06EAF(WORK,NRIFL2,IFAIL)
      NRIP1=NRIFL+1
      WORK(1)=WORK(1)*FK(IFK+1)
      WORK(NRIP1)=WORK(NRIP1)*FK(IFK+NRIP1)
      DO 75 I=2,NRIFL
      N2I=NRIFL2-I+2
      SAFE=WORK(I)
      WORK(I)=SAFE*FK(IFK+I)-WORK(N2I)*FK(IFK+N2I)
   75 WORK(N2I)=-SAFE*FK(IFK+N2I)-WORK(N2I)*FK(IFK+I)
      CALL C06EBF(WORK,NRIFL2,IFAIL)
      DO 77 J=1,NRIFL
   77 Y(KSPOS+NR+J)=Y(KSPOS+NR+J)+WORK(NRIFL+J-1)
   80 CONTINUE
      RETURN
      END
C
      SUBROUTINE VALINT(N,X,F,IFLAG)
C --- NON-LINEAR SYSTEM FOR INITIAL VALUES ---
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(N),F(N),GS(6)
      COMMON /SYST/A(6,6),H,B(6)
      DO 10 I=1,6
   10 GS(I)=GSY(I*H,X(I))
      DO 20 I=1,6
      SUM=X(I)-B(I)
C --- FOR FIRST KIND EQUATIONS EXCHANGE THE ABOVE STATEMENT BY
C     SUM=-B(I)
      DO 15 J=1,6
   15 SUM=SUM-A(I,J)*GS(J)
   20 F(I)=SUM
      RETURN
      END
C
      DOUBLE PRECISION FUNCTION STEP(X)
C --- NON-LINEAR EQUATION FOR FORWARD INTEGRATION ---
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON /FORWAR/C1,F1,T1
      STEP=X-C1*GSY(T1,X)-F1
C --- FOR FIRST KIND EQUATIONS EXCHANGE THE ABOVE STATEMENT BY
C     STEP=-C1*GSY(T1,X)-F1
      RETURN
      END
C
      SUBROUTINE WBDF4(W,WS,NWS,X,Y,NS)
C --- WEIGHTS FOR THE QUADRATURE METHOD BDF4**1/2
C --- TOGETHER WITH THE WEIGHTS OF THE STARTING QUADRATURE
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION W(1),WS(NWS,1),X(1),Y(1),P(5),S(7),A(7,7)
C --- POLYNOMIAL RHO
      K=4
      P(1)=25.D0/12.D0
      P(2)=-4.D0
      P(3)=3.D0
      P(4)=-4.D0/3.D0
      P(5)=1.D0/4.D0
      KP1=K+1
      KP2=K+2
C
C --- COMPUTE THE WEIGHTS W(I) BY NEWTON'S METHOD: W=3/2*W-1/2*P*W**3
C
      N=1
C --- COMPUTE W(1)
```

```
      W(1)=1.D0/DSQRT(P(1))
C --- COMPUTE P=-0.5*P
      DO 3 J=1,KP1
    3 P(J)=-0.5D0*P(J)
C --- DOUBLE N AND FILL UP W WITH 0
    5 CONTINUE
      NH=N
      N=2*NH
      DO 7 I=NH+1,N
    7 W(I)=0.D0
C --- COMPUTE Y=W*P
      NN=MIN0(NH+K,N)
      DO 12 I=1,NN
      SUM=0.D0
      KK=MIN0(KP1,I)
      DO 10 J=1,KK
   10 SUM=SUM+W(I-J+1)*P(J)
   12 Y(I)=SUM
      DO 15 I=NN+1,N
   15 Y(I)=0.D0
C --- COMPUTE Y=W*Y
      DO 18 I=1,N
   18 X(I)=W(I)
      CALL CONVOL(X,Y,N)
C --- COMPUTE ONCE MORE Y=W*Y
      DO 20 I=1,N
   20 X(I)=W(I)
      CALL CONVOL(X,Y,N)
C --- STORE NEW WEIGHTS
      DO 25 I=NH+1,N
   25 W(I)=Y(I)
C --- REPEAT UNTIL ALL WEIGHTS ARE COMPUTED
      IF (N.LT.NS) GOTO 5
C
C --- COMPUTE THE STARTING WEIGHTS WS(J,N)
C
      NB=7
C --- EXACT SEMI-INTEGRALS FOR 1,T**1/2,T,T**3/2,T**2,T**5/2,T**3
      WPI=DSQRT(3.14159265358979323 84D0)
      S(1)=2.D0/WPI
      S(2)=WPI*0.5D0
      S(3)=4.D0/(3.D0*WPI)
      S(4)=WPI*0.375D0
      S(5)=16.D0/(15.D0*WPI)
      S(6)=WPI*0.3125D0
      S(7)=S(5)*6.D0/7.D0
C --- RIGHT-HAND SIDES OF THE LINEAR SYSTEM
      DO 45 J=1,NB
      B=(J-1)/2.D0
      BB=B+0.5D0
      DO 30 N=1,NS
      Y(N)=(N+NB-1.D0)**B
   30 X(N)=W(N)
      CALL CONVOL(X,Y,NS)
      DO 40 N=1,NB-1
   40 WS(J,N)=S(J)*N**BB
      DO 45 N=NB,NB+NS-1
   45 WS(J,N)=S(J)*N**BB-Y(N-NB+1)
C --- THE MATRIX
      DO 50 J=1,NB
   50 A(1,J)=1.D0
      DO 60 I=2,NB
      B=(I-1.D0)/2.D0
      DO 60 J=1,NB
```

```
   60   A(I,J)=(J-1.D0)**B
C --- SOLVE THE LINEAR SYSTEM FOR NS+NB-1 RIGHT-HAND SIDES
        IFAIL=0
        CALL F03AFF(NB,X02AAF(XXXX),A,NB,D1,ID,X,IFAIL)
        CALL F04AJF(NB,NS+NB-1,A,NB,X,WS,NB)
        RETURN
        END
C
        SUBROUTINE CONVOL(X,Y,N)
        IMPLICIT REAL*8 (A-H,O-Z)
        DIMENSION X(1),Y(1)
        NT2=N*2
        SQN=DSQRT(DFLOAT(NT2))
        DO 10 I=1,N
        X(I)=X(I)*SQN
        IN=I+N
        X(IN)=0.D0
   10   Y(IN)=0.D0
        IFAIL=0
        CALL C06EAF(X,NT2,IFAIL)
        CALL C06EAF(Y,NT2,IFAIL)
        NP1=N+1
        Y(1)=Y(1)*X(1)
        Y(NP1)=Y(NP1)*X(NP1)
        DO 15 I=2,N
        N2I=NT2-I+2
        SAFE=Y(I)
        Y(I)=SAFE*X(I)-Y(N2I)*X(N2I)
   15   Y(N2I)=-SAFE*X(N2I)-Y(N2I)*X(I)
        CALL C06EBF(Y,NT2,IFAIL)
        RETURN
        END
C
C
C  + + + + + + + + + + + + + + + + + + +
C --- DRIVER FOR SUBROUTINE ABELFT
C  + + + + + + + + + + + + + + + + + + +
C
        IMPLICIT REAL*8 (A-H,O-Z)
        PARAMETER (M=9,NWS=7,N=2**M,NT2=2*N,N6=N+6,N7=N+7)
        REAL*8 Y(N7),FK(NT2),WORK(NT2),W(N),WS(NWS,N6)
        XEND=8.D0
C
        CALL ABELFT(M,XEND,Y,FK,WORK,W,WS,NWS)
C
        H=XEND/N6
        DO 3 I=1,N6,50
    3   WRITE (6,*) ' X = ',(I-1)*H,'     Y = ',Y(I)
        STOP
        END
C
        DOUBLE PRECISION FUNCTION GSY(S,Y)
        IMPLICIT REAL*8 (A-H,O-Z)
        GSY=-(Y-DSIN(S))**3
        RETURN
        END
C
        DOUBLE PRECISION FUNCTION FCN(X)
        IMPLICIT REAL*8 (A-H,O-Z)
        FCN=0.D0
        RETURN
        END
C
        DOUBLE PRECISION FUNCTION FKER(X)
        IMPLICIT REAL*8 (A-H,O-Z)
        FKER=1.D0
        RETURN
        END
```